Package: LGDtoolkit (via r-universe)

August 23, 2024

Title Collection of Tools for LGD Rating Model Development

Version 0.2.0

Maintainer Andrija Djurovic <`djandrija@gmail.com`>

Description The goal of this package is to cover the most common steps in Loss Given Default (LGD) rating model development. The main procedures available are those that refer to bivariate and multivariate analysis. In particular two statistical methods for multivariate analysis are currently implemented – OLS regression and fractional logistic regression. Both methods are also available within different blockwise model designs and both have customized stepwise algorithms. Descriptions of these customized designs are available in Siddiqi (2016) <doi:10.1002/9781119282396.ch10> and Anderson, R.A. (2021) <doi:10.1093/oso/9780192844194.001.0001>. Although they are explained for PD model, the same designs are applicable for LGD model with different underlying regression methods (OLS and fractional logistic regression). To cover other important steps for LGD model development, it is recommended to use 'LGDtoolkit' package along with 'PDtoolkit', and 'monobin' (or 'monobinShiny') packages. Additionally, 'LGDtoolkit' provides set of procedures handy for initial and periodical model validation.

License GPL (>= 3)

URL `https://github.com/andrija-djurovic/LGDtoolkit`

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

Depends R (>= 2.10)

Imports dplyr, monobin

Repository https://andrija-djurovic.r-universe.dev

RemoteUrl https://github.com/andrija-djurovic/lgdtoolkit

RemoteRef HEAD

RemoteSha 1481c9b8cf6e9ed81718327f1acd3a70be51acd9

Contents

`embedded.blocks` *Embedded blocks regression*

Description

`embedded.blocks` performs blockwise regression where the predictions of each blocks' model is used as an risk factor for the model of the following block.

Usage

```
embedded.blocks(method, target, db, blocks, reg.type = "ols", p.value = 0.05)
```

Arguments

`method`	Regression method applied on each block. Available methods: `"stepFWD"` or `"stepRPC"`.
`target`	Name of target variable within `db` argument.
`db`	Modeling data with risk factors and target variable.
`blocks`	Data frame with defined risk factor groups. It has to contain the following columns: `rf` and `block`.
`reg.type`	Regression type. Available options are: `"ols"` for OLS regression and `"frac.logit"` for fractional logistic regression. Default is `"ols"`. For `"frac.logit"` option, target has to have all values between 0 and 1.

`p.value`	Significance level of p-value for the estimated coefficient. For numerical risk factors this value is is directly compared to p-value of the estimated coefficient, while for categorical multiple Wald test is employed and its p-value is used for comparison with selected threshold (`p.value`).

Value

The command `embedded.blocks` returns a list of three objects.
The first object (`model`) is the list of the models of each block (an object of class inheriting from `"lm"`).
The second object (`steps`), is the data frame with risk factors selected from the each block.
The third object (`dev.db`), returns the list of block's model development databases.

See Also

`staged.blocks`, `ensemble.blocks`, `stepFWD` and `stepRPC`.

Examples

```
library(monobin)
library(LGDtoolkit)
data(lgd.ds.c)
#stepwise with discretized risk factors
#same procedure can be run on continuous risk factors and mixed risk factor types
num.rf <- sapply(lgd.ds.c, is.numeric)
num.rf <- names(num.rf)[!names(num.rf)%in%"lgd" & num.rf]
num.rf
for (i in 1:length(num.rf)) {
num.rf.l <- num.rf[i]
lgd.ds.c[, num.rf.l] <- sts.bin(x = lgd.ds.c[, num.rf.l], y = lgd.ds.c[, "lgd"])[[2]]
}
str(lgd.ds.c)
set.seed(321)
blocks <- data.frame(rf = names(lgd.ds.c)[!names(lgd.ds.c)%in%"lgd"],
     block = sample(1:3, ncol(lgd.ds.c) - 1, rep = TRUE))
blocks <- blocks[order(blocks$block, blocks$rf), ]
lgd.ds.c$lgd[lgd.ds.c$lgd > 1] <- 1
res <- LGDtoolkit::embedded.blocks(method = "stepRPC",
       target = "lgd",
       db = lgd.ds.c,
       blocks = blocks,
       reg.type = "frac.logit",
       p.value = 0.05)
names(res)
res$models
summary(res$models[[3]])
```

`ensemble.blocks` *Ensemble blocks regression*

Description

`ensemble.blocks` performs blockwise regression where the predictions of each blocks' model are integrated into a final model. The final model is estimated in the form of OLS or fractional logistic regression regression without any check of the estimated coefficients (e.g. statistical significance or sign of the estimated coefficients).

Usage

```
ensemble.blocks(method, target, db, blocks, reg.type = "ols", p.value = 0.05)
```

Arguments

`method`	Regression method applied on each block. Available methods: `"stepFWD"` or `"stepRPC"`.
`target`	Name of target variable within `db` argument.
`db`	Modeling data with risk factors and target variable.
`blocks`	Data frame with defined risk factor groups. It has to contain the following columns: `rf` and `block`.
`reg.type`	Regression type. Available options are: `"ols"` for OLS regression and `"frac.logit"` for fractional logistic regression. Default is `"ols"`. For `"frac.logit"` option, target has to have all values between 0 and 1.
`p.value`	Significance level of p-value for the estimated coefficient. For numerical risk factors this value is is directly compared to p-value of the estimated coefficient, while for categorical multiple Wald test is employed and its p-value is used for comparison with selected threshold (`p.value`).

Value

The command `embeded.blocks` returns a list of three objects.
The first object (`model`) is the list of the models of each block (an object of class inheriting from `"lm"`).
The second object (`steps`), is the data frame with risk factors selected from the each block.
The third object (`dev.db`), returns the list of block's model development databases.

See Also

`staged.blocks`, `embedded.blocks`, `stepFWD` and `stepRPC`.

Examples

```
library(monobin)
library(LGDtoolkit)
data(lgd.ds.c)
#stepwise with discretized risk factors
#same procedure can be run on continuous risk factors and mixed risk factor types
num.rf <- sapply(lgd.ds.c, is.numeric)
num.rf <- names(num.rf)[!names(num.rf)%in%"lgd" & num.rf]
num.rf
for (i in 1:length(num.rf)) {
num.rf.l <- num.rf[i]
lgd.ds.c[, num.rf.l] <- sts.bin(x = lgd.ds.c[, num.rf.l], y = lgd.ds.c[, "lgd"])[[2]]
}
str(lgd.ds.c)
set.seed(2211)
blocks <- data.frame(rf = names(lgd.ds.c)[!names(lgd.ds.c)%in%"lgd"],
     block = sample(1:3, ncol(lgd.ds.c) - 1, rep = TRUE))
blocks <- blocks[order(blocks$block, blocks$rf), ]
res <- LGDtoolkit::ensemble.blocks(method = "stepFWD",
       target = "lgd",
       db = lgd.ds.c,
       blocks = blocks,
       reg.type = "ols",
       p.value = 0.05)
names(res)
res$models
summary(res$models[[4]])
```

`heterogeneity` *Testing heterogeneity of the LGD rating model*

Description

`heterogeneity` performs heterogeneity testing of LGD model based on the rating pools. This test is usually applied on application portfolio, but it can be applied also on model development sample.

Usage

```
heterogeneity(app.port, loss, pools, method = "t.test", alpha = 0.05)
```

Arguments

`app.port`	Application portfolio (data frame) which contains realized loss (LGD) values and LGD pools in use.
`loss`	Name of the column that represents realized loss (LGD).
`pools`	Name of the column that represents LGD pools.
`method`	Statistical test. Available options are `t.test` (default) and `wilcox.test`.
`alpha`	Significance level of statistical test. Default is 0.05.

Details

Testing procedure starts with summarizing the number of observations and average loss per LGD pool. After that statistical test is applied on adjacent rating grades. Testing hypothesis is that average realized loss of pool `i` is less or greater than average realized loss of pools `i - 1`, where `i` takes the values from 2 to the number of unique pools. Direction of alternative hypothesis (less or greater) is determined automatically based on correlation direction of realized average loss per pool. Incomplete cases, identified based on realized loss (`loss`) and rating pool (`pools`) columns are excluded from the summary table and testing procedure. If identified, warning will be returned.

Value

The command `heterogeneity` returns a data frame with the following columns:

- pool: Unique values of pool from application portfolio.
- no: Number of complete observations.
- mean: Average realized loss.
- alpha: Selected significance level
- p.val: Test p-value.
- res: Accepted hypothesis.

Examples

```
library(monobin)
library(LGDtoolkit)
data(lgd.ds.c)
#build dummy model
rf <- c("rf_02", "rf_01", "rf_16", "rf_03", "rf_09")
for   (i in 1:length(rf)) {
     rf_l <- rf[i]
     lgd.ds.c[, rf_l] <- sts.bin(x = lgd.ds.c[, rf_l],
                                 y = lgd.ds.c[, "lgd"])[[2]]
     }
str(lgd.ds.c)
frm <- paste0("lgd ~ ", paste(rf, collapse = " + "))
model <- lm(formula = as.formula(frm), data = lgd.ds.c)
summary(model)$coefficients
summary(model)$r.squared
#create lgd pools
lgd.ds.c$pred <- unname(predict(model))
lgd.ds.c$pool <- sts.bin(x = lgd.ds.c$pred,
                        y = lgd.ds.c$lgd)[[2]]
#create dummy application portfolio
set.seed(642)
app.port <- lgd.ds.c[sample(1:nrow(lgd.ds.c), 500, replace = FALSE), ]
#simulate realized lgd values
app.port$lgd.r <- app.port$lgd
#test heterogeneity
heterogeneity(app.port = app.port,
   loss = "lgd.r",
   pools = "pool",
             method = "t.test",
             alpha = 0.05)
```

`homogeneity` *Testing homogeneity of the LGD rating model*

Description

`homogeneity` performs homogeneity testing of LGD model based on the rating pools and selected segment. This test is usually applied on application portfolio, but it can be applied also on model development sample. Additionally, this method requires higher number of observations per segment modalities within each rating in order to produce available results. For segments with less than 30 observations, test is not performed.

Usage

```
homogeneity(
  app.port,
  loss,
  pools,
  segment,
  segment.num,
  method = "t.test",
  alpha = 0.05
)
```

Arguments

`app.port`	Application portfolio (data frame) which contains at lease realized loss (LGD), pools in use and variable used as a segment.
`loss`	Name of the column that represents realized loss (LGD).
`pools`	Name of the column that represents LGD pools.
`segment`	Name of the column that represent testing segments. If it is of numeric type, than it is first grouped into `segment.num` of groups otherwise is it used as supplied.
`segment.num`	Number of groups used for numeric variables supplied as a segment. Only applicable if `segment` is of numeric type.
`method`	Statistical test. Available options are `t.test` (default) and `wilcox.test`.
`alpha`	Significance level of statistical test. Default is 0.05.

Details

Testing procedure is implemented for each rating separately comparing average realized loss from one segment modality to the average realized loss from the rest of segment modalities.

Value

The command `homogeneity` returns a data frame with the following columns:

- segment.var: Variable used as a segment.
- pool: Unique values of pools from application portfolio..
- segment.mod: Tested segment modality. Average realized loss from this segment is compared with average realized loss from the rest of the modalities within the each rating.
- no: Number of observations in the analyzed pool.
- avg: Average realized loss in the analyzed pool.
- avg.segment: Average realized loss per analyzed segment modality within certain pool.
- avg.rest: Average realized loss of the rest of segment modalities within certain pool.
- no.segment: Number of observations of the analyzed segment modality.
- no.rest: Number of observations of the rest of the segment modalities.
- p.val: Two proportion test (two sided) p-value.
- alpha: Selected significance level.
- res: Accepted hypothesis.

Examples

```
library(monobin)
library(LGDtoolkit)
data(lgd.ds.c)
#build dummy model
rf <- c("rf_01", "rf_02", "rf_16", "rf_03", "rf_09")
for   (i in 1:length(rf)) {
     rf_l <- rf[i]
     lgd.ds.c[, rf_l] <- sts.bin(x = lgd.ds.c[, rf_l],
                                 y = lgd.ds.c[, "lgd"])[[2]]
     }
str(lgd.ds.c)
frm <- paste0("lgd ~ ", paste(rf, collapse = " + "))
model <- lm(formula = as.formula(frm), data = lgd.ds.c)
summary(model)$coefficients
#create lgd pools
lgd.ds.c$pred <- unname(predict(model))
lgd.ds.c$pool <- sts.bin(x = lgd.ds.c$pred,
                         y = lgd.ds.c$lgd)[[2]]
#test homogeneity on development sample
#(the same procedure can be applied on application portfolio)
homogeneity(app.port = lgd.ds.c,
           loss = "lgd",
           pools = "pool",
           segment = "rf_03",
           segment.num = 3,
           method = "t.test",
           alpha = 0.05)
```

`interaction.transformer`
Extract risk factors interaction from decision tree

Description

`interaction.transformer` extracts the interaction between supplied risk factors from decision tree. It implements customized decision tree algorithm that takes into account different conditions such as minimum percentage of observations and defaults in each node, maximum tree depth and monotonicity condition at each splitting node. Sum of squared errors is used as metric for node splitting .

Usage

```
interaction.transformer(
  db,
  rf,
  target,
  min.pct.obs,
  min.avg.rate,
  max.depth,
  monotonicity,
  create.interaction.rf
)
```

Arguments

`db`	Data frame of risk factors and target variable supplied for interaction extraction.
`rf`	Character vector of risk factor names on which decision tree is run.
`target`	Name of target variable within db argument.
`min.pct.obs`	Minimum percentage of observation in each leaf.
`min.avg.rate`	Minimum average target rate in each leaf..
`max.depth`	Maximum number of splits.
`monotonicity`	Logical indicator. If `TRUE`, observed trend between risk factor and target will be preserved in splitting node.
`create.interaction.rf`	Logical indicator. If `TRUE`, second element of the output will be data frame with interaction modalities.

Value

The command `interaction.transformer` returns a list of two data frames. The first data frame provides the tree summary. The second data frame is a new risk factor extracted from decision tree.

Examples

```
library(LGDtoolkit)
data(lgd.ds.c)
it <- LGDtoolkit::interaction.transformer(db = lgd.ds.c,
                rf = c("rf_01", "rf_03"),
                             target = "lgd",
                             min.pct.obs = 0.05,
                             min.avg.rate = 0.01,
                             max.depth = 2,
                             monotonicity = TRUE,
                             create.interaction.rf = TRUE)
names(it)
it[["tree.info"]]
tail(it[["interaction"]])
table(it[["interaction"]][, "rf.inter"], useNA = "always")
```

`kfold.idx` *Indices for K-fold validation*

Description

`kfold.idx` provides indices for K-fold validation.

Usage

```
kfold.idx(target, k = 10, type, num.strata = 4, seed = 2191)
```

Arguments

`target`	Continuous target variable.
`k`	Number of folds. If `k` is equal or greater than the number of observations of target variable, then validation procedure is equivalent to leave one out cross-validation (LOOCV) method. Default is set to 10.
`type`	Sampling type. Possible options are `"random"` and `"stratified"`.
`num.strata`	Number of strata for `"stratified"` type. Default is 4.
`seed`	Random seed needed for ensuring the result reproducibility. Default is 2191.

Value

The command `kfold.idx` returns a list of k folds estimation and validation indices.

Examples

```
library(monobin)
library(LGDtoolkit)
data(lgd.ds.c)
#random k-folds
kf.r <- LGDtoolkit::kfold.idx(target = lgd.ds.c$lgd, k = 5,
   type = "random", seed = 2211)
sapply(kf.r, function(x) c(mean(lgd.ds.c$lgd[x[[1]]]), mean(lgd.ds.c$lgd[x[[2]]])))
sapply(kf.r, function(x) length(x[[2]]))
#stratified k-folds
kf.s <- LGDtoolkit::kfold.idx(target = lgd.ds.c$lgd, k = 5,
                              type = "stratified", num.strata = 10, seed = 2211)
sapply(kf.s, function(x) c(mean(lgd.ds.c$lgd[x[[1]]]), mean(lgd.ds.c$lgd[x[[2]]])))
sapply(kf.s, function(x) length(x[[2]]))
```

`kfold.vld` *K-fold model cross-validation*

Description

`kfold.vld` performs k-fold model cross-validation. The main goal of this procedure is to generate main model performance metrics such as absolute mean square error, root mean square error or R-squared based on resampling method. Note that functions' argument model accepts `"lm"` and `"glm"` class but for `"glm"` only `"quasibinomial("logit")"` family will be considered.

Usage

```
kfold.vld(model, k = 10, seed = 1984)
```

Arguments

`model`	Model in use, an object of class inheriting from `"lm"`
`k`	Number of folds. If `k` is equal or greater than the number of observations of modeling data frame, then validation procedure is equivalent to leave one out cross-validation (LOOCV) method. For LOOCV, R-squared is not calculated. Default is set to 10.
`seed`	Random seed needed for ensuring the result reproducibility. Default is 1984.

Value

The command `kfold.vld` returns a list of two objects.
The first object (`iter`), returns iteration performance metrics.
The second object (`summary`), is the data frame of iterations averages of performance metrics.

Examples

```
library(monobin)
library(LGDtoolkit)
data(lgd.ds.c)
#discretized some risk factors
num.rf <- c("rf_01", "rf_02", "rf_03", "rf_09", "rf_16")
for (i in 1:length(num.rf)) {
num.rf.l <- num.rf[i]
lgd.ds.c[, num.rf.l] <- sts.bin(x = lgd.ds.c[, num.rf.l], y = lgd.ds.c[, "lgd"])[[2]]
}
str(lgd.ds.c)
#run linear regression model
reg.mod.1 <- lm(lgd ~ ., data = lgd.ds.c[, c(num.rf, "lgd")])
summary(reg.mod.1)$coefficients
#perform k-fold validation
LGDtoolkit::kfold.vld(model = reg.mod.1 , k = 10, seed = 1984)
#run fractional logistic regression model
lgd.ds.c$lgd[lgd.ds.c$lgd > 1] <- 1
reg.mod.2 <- glm(lgd ~ ., family = quasibinomial("logit"), data = lgd.ds.c[, c(num.rf, "lgd")])
summary(reg.mod.2)$coefficients
LGDtoolkit::kfold.vld(model = reg.mod.2 , k = 10, seed = 1984)
```

`lgd.ds.c` *Synthetic modeling dataset*

Description

Synthetic modeling dataset of observed LGD values for contracts with complete recovery process. Dataset consists of 1200 observations and 19 risk factors.

Usage

```
lgd.ds.c
```

Format

An object of class `data.frame` with 1200 rows and 20 columns.

`r.squared` *Coefficient of determination*

Description

`r.squared` returns coefficient of determination for risk factors supplied in data frame db. Implemented algorithm processes numerical as well as categorical risk factor.
Usually, this procedure is applied as starting point of bivariate analysis in LGD model development.

Usage

```
r.squared(db, target)
```

Arguments

`db`	Data frame of risk factors and target variable supplied for bivariate analysis.
`target`	Name of target variable within db argument.

Value

The command `r.squared` returns the data frames with a following statistics: name of the processed risk factor (`rf`), type of processed risk factor (`rf.type`), number of missing and infinite observations (`miss.inf`), percentage of missing and infinite observations (`miss.inf.pct`), coefficient of determination (`r.squared`)

Examples

```
library(monobin)
library(LGDtoolkit)
data(lgd.ds.c)
r.squared(db = lgd.ds.c, target = "lgd")
#add categorical risk factor
lgd.ds.c$rf_03_bin <- sts.bin(x = lgd.ds.c$rf_03, y = lgd.ds.c$lgd)[[2]]
r.squared(db = lgd.ds.c, target = "lgd")
#add risk factor with all missing, only one complete case and zero variance risk factor
lgd.ds.c$rf_20 <- NA
lgd.ds.c$rf_21 <- c(1, rep(NA, nrow(lgd.ds.c) - 1))
lgd.ds.c$rf_22 <- c(c(1, 1), rep(NA, nrow(lgd.ds.c) - 2))
r.squared(db = lgd.ds.c, target = "lgd")
```

`rf.interaction.transformer` *Extract interactions from random forest*

Description

`rf.interaction.transformer` extracts the interactions from random forest. It implements customized random forest algorithm that takes into account different conditions (for single decision tree) such as minimum percentage of observations and defaults in each node, maximum tree depth and monotonicity condition at each splitting node. Sum of squared errors index is used as metric for node splitting .

Usage

```
rf.interaction.transformer(
  db,
  rf,
  target,
  num.rf = NA,
  num.tree,
  min.pct.obs,
  min.avg.rate,
  max.depth,
  monotonicity,
  create.interaction.rf,
  seed = 991
)
```

Arguments

db	Data frame of risk factors and target variable supplied for interaction extraction.
rf	Character vector of risk factor names on which decision tree is run.
target	Name of target variable within db argument.
num.rf	Number of risk factors randomly selected for each decision tree. If default value (NA) is supplied, then number of risk factors will be calculated as sqrt(number of all supplied risk factors).
num.tree	Number of decision trees used for random forest.
min.pct.obs	Minimum percentage of observation in each leaf.
min.avg.rate	Minimum average target rate in each leaf.
max.depth	Maximum number of splits.
monotonicity	Logical indicator. If TRUE, observed trend between risk factor and target will be preserved in splitting node.
create.interaction.rf	Logical indicator. If TRUE, second element of the output will be data frame with interaction modalities.
seed	Random seed to ensure result reproducibility. Default is 991.

Value

The command rf.interaction.transformer returns a list of two data frames. The first data frame provides the trees summary. The second data frame is a new risk factor extracted from random forest.

Examples

```
library(LGDtoolkit)
data(lgd.ds.c)
rf.it <- LGDtoolkit::rf.interaction.transformer(db = lgd.ds.c,
      rf = names(lgd.ds.c)[!names(lgd.ds.c)%in%"lgd"],
      target = "lgd",
      num.rf = NA,
      num.tree = 3,
      min.pct.obs = 0.05,
      min.avg.rate = 0.01,
      max.depth = 2,
        monotonicity = TRUE,
        create.interaction.rf = TRUE,
        seed = 789)
names(rf.it)
rf.it[["tree.info"]]
tail(rf.it[["interaction"]])
table(rf.it[["interaction"]][, 1], useNA = "always")
```

sc.merge *Special case merging procedure*

Description

sc.merge performs procedure of merging special case bins with one from complete cases. This procedure can be used not only for LGD model development, but also for PD and EAD, i.e. for all models that have categorical risk factors.

Usage

```
sc.merge(x, y, sc = "SC", sc.merge = "closest", force.trend = "modalities")
```

Arguments

x	Categorical risk factor.
y	Target variable.
sc	Vector of special case values. Default is set to "SC".
sc.merge	Merging method. Available options are: "first", "last" and "closest". Default value is "closest" and it is determined as the bin with the closest average target rate.
force.trend	Defines how initial summary table will be ordered. Possible options are: "modalities" and "y.avg". If "modalities" is selected, then merging will be performed forward based on alphabetic order of risk factor modalities. On the other hand, if "y.avg" is selected, then bins merging will be performed forward based on increasing order of mean of target variable per modality.

Value

The command sc.merge generates a list of two objects. The first object, data frame summary.tbl presents a summary table of final binning, while x.trans is a vector of recoded values.

Examples

```
library(monobin)
library(LGDtoolkit)
data(lgd.ds.c)
rf.03.bin.s <- sts.bin(x = lgd.ds.c$rf_03, y = lgd.ds.c$lgd)
rf.03.bin.s[[1]]
```

```
table(rf.03.bin.s[[2]])
lgd.ds.c$rf_03_bin <- rf.03.bin.s[[2]]
rf.03.bin.c <- sc.merge(x = lgd.ds.c$rf_03_bin,
   y = lgd.ds.c$lgd,
   sc = "SC",
   sc.merge = "closest",
   force.trend = "modalities")
str(rf.03.bin.c)
rf.03.bin.c[[1]]
table(rf.03.bin.c[[2]])
```

staged.blocks *Staged blocks regression*

Description

staged.blocks performs blockwise regression where the predictions of each blocks' model is used as an offset for the model of the following block.

Usage

```
staged.blocks(method, target, db, blocks, reg.type = "ols", p.value = 0.05)
```

Arguments

method	Regression method applied on each block. Available methods: "stepFWD" or "stepRPC".
target	Name of target variable within db argument.
db	Modeling data with risk factors and target variable.
blocks	Data frame with defined risk factor groups. It has to contain the following columns: rf and block.
reg.type	Regression type. Available options are: "ols" for OLS regression and "frac.logit" for fractional logistic regression. Default is "ols". For "frac.logit" option, target has to have all values between 0 and 1.
p.value	Significance level of p-value for the estimated coefficient. For numerical risk factors this value is is directly compared to p-value of the estimated coefficient, while for categorical multiple Wald test is employed and its p-value is used for comparison with selected threshold (p.value).

Value

The command staged.blocks returns a list of three objects.
The first object (model) is the list of the models of each block (an object of class inheriting from "lm").
The second object (steps), is the data frame with risk factors selected from the each block.
The third object (dev.db), returns the list of block's model development databases.

See Also

`embedded.blocks`, `ensemble.blocks`, `stepFWD` and `stepRPC`.

Examples

```
library(LGDtoolkit)
data(lgd.ds.c)
#stepwise with continuous risk factors
set.seed(123)
blocks <- data.frame(rf = names(lgd.ds.c)[!names(lgd.ds.c)%in%"lgd"],
     block = sample(1:3, ncol(lgd.ds.c) - 1, rep = TRUE))
blocks <- blocks[order(blocks$block, blocks$rf), ]
res <- LGDtoolkit::staged.blocks(method = "stepFWD",
     target = "lgd",
     db = lgd.ds.c,
     reg.type = "ols",
     blocks = blocks,
     p.value = 0.05)
names(res)
res$models
summary(res$models[[3]])
identical(unname(predict(res$models[[1]], newdata = res$dev.db[[1]])),
    res$dev.db[[2]]$offset.vals)
```

`stepFWD` *Customized stepwise (OLS & fractional logistic) regression with p-value and trend check*

Description

`stepFWD` customized stepwise regression with p-value and trend check. Trend check is performed comparing observed trend between target and analyzed risk factor and trend of the estimated coefficients within the linear regression. Note that procedure checks the column names of supplied `db` data frame therefore some renaming (replacement of special characters) is possible to happen. For details check help example.

Usage

```
stepFWD(
  start.model,
  p.value = 0.05,
  db,
  reg.type = "ols",
  check.start.model = TRUE,
  offset.vals = NULL
)
```

Arguments

start.model	Formula class that represents starting model. It can include some risk factors, but it can be defined only with intercept (y ~ 1 where y is target variable).
p.value	Significance level of p-value of the estimated coefficients. For numerical risk factors this value is is directly compared to the p-value of the estimated coefficients, while for categorical risk factors multiple Wald test is employed and its p-value is used for comparison with selected threshold (p.value).
db	Modeling data with risk factors and target variable. Risk factors can be categorized or continuous.
reg.type	Regression type. Available options are: "ols" for OLS regression and "frac.logit" for fractional logistic regression. Default is "ols". For "frac.logit" option, target has to have all values between 0 and 1.
check.start.model	Logical (TRUE or FALSE), if risk factors from the starting model should be checked for p-value and trend in stepwise process. Default is TRUE.
offset.vals	This can be used to specify an a priori known component to be included in the linear predictor during fitting. This should be NULL or a numeric vector of length equal to the number of cases. Default is NULL.

Value

The command stepFWD returns a list of four objects.
The first object (model), is the final model, an object of class inheriting from "glm".
The second object (steps), is the data frame with risk factors selected at each iteration.
The third object (warnings), is the data frame with warnings if any observed. The warnings refer to the following checks: if risk factor has more than 10 modalities or if any of the bins (groups) has less than 5% of observations.
The final, fourth, object dev.db returns the model development database.

Examples

```
library(monobin)
library(LGDtoolkit)
data(lgd.ds.c)
#stepwise with discretized risk factors
#same procedure can be run on continuous risk factors and mixed risk factor types
num.rf <- sapply(lgd.ds.c, is.numeric)
num.rf <- names(num.rf)[!names(num.rf)%in%"lgd" & num.rf]
num.rf
#select subset of numerical risk factors
num.rf <- num.rf[1:10]
for (i in 1:length(num.rf)) {
num.rf.l <- num.rf[i]
lgd.ds.c[, num.rf.l] <- sts.bin(x = lgd.ds.c[, num.rf.l], y = lgd.ds.c[, "lgd"])[[2]]
}
str(lgd.ds.c)
res <- LGDtoolkit::stepFWD(start.model = lgd ~ 1,
    p.value = 0.05,
    db = lgd.ds.c[, c(num.rf, "lgd")],
    reg.type = "ols")
names(res)
summary(res$model)$coefficients
res$steps
summary(res$model)$r.squared
```

`stepRPC` *Stepwise (OLS & fractional logistic) regression based on risk profile concept*

Description

`stepRPC` customized stepwise regression with p-value and trend check which additionally takes into account the order of supplied risk factors per group when selects a candidate for the final regression model. Trend check is performed comparing observed trend between target and analyzed risk factor and trend of the estimated coefficients. Note that procedure checks the column names of supplied `db` data frame therefore some renaming (replacement of special characters) is possible to happen. For details, please, check the help example.

Usage

```
stepRPC(
  start.model,
  risk.profile,
  p.value = 0.05,
  db,
  reg.type = "ols",
  check.start.model = TRUE,
  offset.vals = NULL
)
```

Arguments

`start.model`	Formula class that represents the starting model. It can include some risk factors, but it can be defined only with intercept (`y ~ 1` where `y` is target variable).
`risk.profile`	Data frame with defined risk profile. It has to contain the following columns: `rf` and `group`. Column `group` defines order of groups that will be tested first as a candidate for the regression model. Risk factors selected in each group are kept as a starting variables for the next group testing. Column `rf` contains all candidate risk factors supplied for testing.
`p.value`	Significance level of p-value of the estimated coefficients. For numerical risk factors this value is is directly compared to the p-value of the estimated coefficients, while for categorical risk factors multiple Wald test is employed and its value is used for comparison with selected threshold (`p.value`).

`db`	Modeling data with risk factors and target variable. All risk factors (apart from the risk factors from the starting model) should be categorized and as of character type.
`reg.type`	Regression type. Available options are: `"ols"` for OLS regression and `"frac.logit"` for fractional logistic regression. Default is `"ols"`. For `"frac.logit"` option, target has to have all values between 0 and 1.
`check.start.model`	Logical (`TRUE` or `FALSE`), if risk factors from the starting model should checked for p-value and trend in stepwise process.
`offset.vals`	This can be used to specify an a priori known component to be included in the linear predictor during fitting. This should be `NULL` or a numeric vector of length equal to the number of cases. Default is `NULL`.

Value

The command `stepRPC` returns a list of four objects.
The first object (`model`), is the final model, an object of class inheriting from `"glm"`.
The second object (`steps`), is the data frame with risk factors selected at each iteration.
The third object (`warnings`), is the data frame with warnings if any observed. The warnings refer to the following checks: if risk factor has more than 10 modalities or if any of the bins (groups) has less than 5% of observations.
The final, fourth, object `dev.db` returns the model development database.

Examples

```
library(monobin)
library(LGDtoolkit)
data(lgd.ds.c)
num.rf <- sapply(lgd.ds.c, is.numeric)
num.rf <- names(num.rf)[!names(num.rf)%in%"lgd" & num.rf]
num.rf
for (i in 1:length(num.rf)) {
num.rf.l <- num.rf[i]
lgd.ds.c[, num.rf.l] <- sts.bin(x = lgd.ds.c[, num.rf.l], y = lgd.ds.c[, "lgd"])[[2]]
}
str(lgd.ds.c)
#define risk factor groups
set.seed(123)
rf.pg <- data.frame(rf = names(lgd.ds.c)[!names(lgd.ds.c)%in%"lgd"],
    group = sample(1:5, ncol(lgd.ds.c) - 1, rep = TRUE))
rf.pg <- rf.pg[order(rf.pg$group, rf.pg$r), ]
rf.pg
res <- LGDtoolkit::stepRPC(start.model = lgd ~ 1,
    risk.profile = rf.pg,
    p.value = 0.05,
    db = lgd.ds.c,
    reg.type = "ols")
names(res)
summary(res$model)$coefficients
summary(res$model)$r.squared
```

Index